Perspectives in Electromagnetics: Creating Simple Graphical User Interfaces for Electromagnetic Codes

Daniel S. Katz

Graphical User Interfaces

Electromagnetic software (and **software** in general) at one time was coded onto a series of cards. These card decks contained both code and input data. When the cards (and job control cards) were fed into a computer, the program was compiled and run, and the results were printed out by a line printer. This was the common method for running programs until the early 1980s, and this method was still being used at some aerospace companies and universities at least as recently as 1990. Around 1970 interactive terminals (that ran at a speed fast enough to be useful) started to appear. These terminals, along with the introduction of disks for storage, allowed users (rather than operators) to type simple commands to run codes, and view the numbers that were the output. Next, programs were written to display the output in a graphical format. Shortly after the introduction of interactive terminals, codes began to be written that allowed users to interact with running jobs. From there, it was a short leap to changing the analysis programs to directly create the graphical output, and to accept graphical input, or at least to be controlled graphically, through a **graphical user interface (GUI)**. However, many codes have not yet reached this stage, and they are still written for batch processing (controlled by typing a command, which does some processing, and creates one or more data files). Sometimes, this is because the analysis takes a long time to run, and is performed in a batch environment (where the series of commands is stored to be a run at a time that is convenient for the computer running the code, not for the user.) Other times, the reasons are purely historical (that is how the code was written , and no one has ever spent any time changing it).

In 1997, a group at JPL decided to create a user-friendly tool for the design and analysis of millimeter-wave instruments ("D. S. Katz, A. Borgioli, T. Cwik, C. Fu, W. A. Imbriale, V. Jamnejad, and P. L. Springer, "A Simple Tool for the Design and Analysis of Multiple-Reflector Antennas in a Multi-Disciplinary Environment," submitted to 1999 IEEE AP-S International Symposium). The main analysis code used by this tool was a physical optics program that had been developed at JPL (W. A. Imbriale and R. E. Hodges, "The Linear-Phase Triangular Facet Approximation in Physical Optics Analysis of Reflector Antennas," *App. Comp. Electromag. Soc. J.*, v. 6, pp. 52-73, 1991). The first focus of our tool was to build a GUI for the analysis code. Our intention was to not modify the Fortran code; simply to write a GUI that would enable us to run the existing code. We decided to use Tcl/Tk for a number of reasons. 1. We were interested in writing a GUI that could be run on multiple platforms (Macintosh, Window, and UNIX), which led us to examine Java/CORBA and Tcl/Tk. 2. We wanted to be able to run the analysis code on a number of supercomputer platforms, and that required that we be able to communicate easily between the platforms. This led us away from Java/CORBA, as we did not have ORBs for all the platforms, and the cost of some of the ORBs was prohibitory. 3. We wanted the GUI to be simple to write, as those of us that were planning to write it were engineers, not GUI designers. Therefore we chose **Tcl/Tk**, and built a package called MOD Tool. A sample window can be seen in figure 1. The material discussed in this column overlaps with MOD Tool, but the goal here is simplicity, rather than creating a full package.

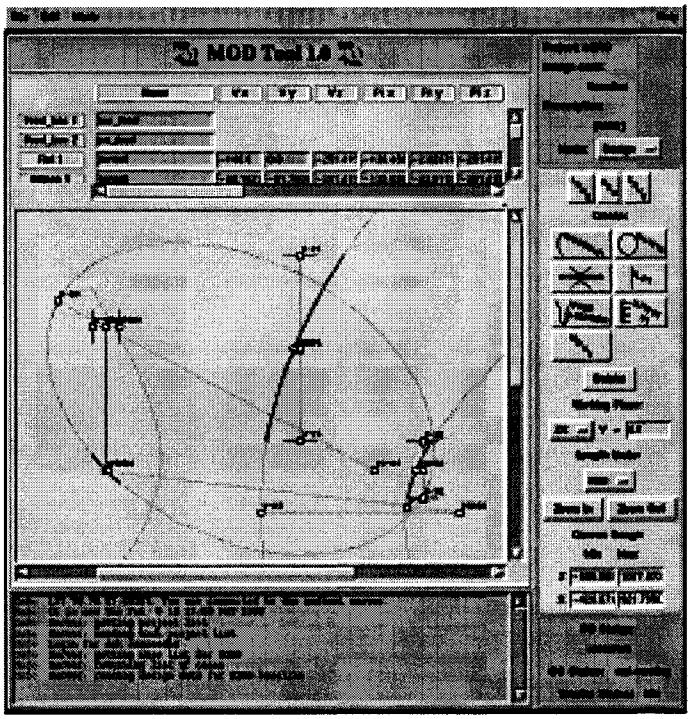

Figure 1. A sample screen from MOD Tool

Introducing Tcl/Tk

One method for writing graphical user interfaces (GUIs) for compiled electromagnetic programs is through the use of **Expectk**. Expectk is built on Tcl/Tk, as will be explained later. The graphics that are used in expect are the same as those used in Tcl/Tk. The ease of creating graphical programs in Tcl/Tk is best shown through the following simple example.

On a system where Tcl/Tk has been installed, the user may run `wish`. `wish` is one of two interpreters that come with Tcl/Tk. It stands for windowing shell. (The other is `tclsh`, a non-graphic Tcl shell.) On a Unix machine, you should type wish, and a small window will appear. On a Mac or PC, double-clicking on the wish icon will start the wish program. Once wish is started, it will either return a prompt `%`, or a console window will appear that has this prompt. Type the following into that window.

The author is with the Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, MS 168-522, Pasadena, CA 91109-8099. E-mail address: d.katz@ieee.org

```
button .b -text "Press Me!" \
  -command {puts "hello world"}
label .l -text "Hello World!"
pack .l .b
```

The first two commands create widgets named `.b` and `.l`, and the third command causes these widgets to be placed on the screen. This will cause a window containing something similar to figure 2 to be generated. This shows that the label and the button are created, and as the reader can discover, pressing the button will cause the words "`hello world`" to print on the screen or the console window. Type "`exit`" in the screen or the console window to exit `wish`. This example is meant to show the simplicity of Tcl/Tk, and the user can experiment with the code to make changes.

Tcl was originally created by John Ousterhout in 1988 as a command language for interactive tools. Tk is a toolkit, based on Tcl, also created while John Ousterhout was at the University of California at Berkeley. Much further development on Tcl/Tk occurred at Sun Microsystems, Inc. Scriptics Corporation was founded in 1998 to continue Tcl/Tk development, after Sun decided that this was not an area it should be directly supporting. The current version of Tcl/Tk is always freely available at the Scriptics web site , listed at the end of this column.

The simplest use of expect is to run some job, to scan the output of that job to find a certain pattern, and to respond to that pattern. This scan-response pair can occur many times, and both the pattern being scanned for and the response being sent can be Tcl/Tk variables. This means that a Tcl/Tk program may be written that will interact with another program. As the Tcl/Tk program can be graphical, this is a simple method for writing a GUI that can control a job, without having to change the underlying job. The user of the GUI only sees the GUI, and not the job. The person that creates the code to run the job does not have to know anything about the GUI, and in fact, the two parts of the interactive code can be written in different periods of time. A GUI writer can create a GUI to run a code without knowing anything about the code, other than how it is run.

A GUI for a Batch Code

Let's now consider an electromagnetic code, such as the physical optics code mentioned previously. This code was written to be run in a batch environment. It reads an input file to determine what geometry to analyze and what outputs to create, then performs the analysis, and writes the output files. This type of code is fairly amenable to being controlled by a GUI, since the GUI simply writes the input file and then runs the code. For simplicity, let's assume that the input file can be used to control the analysis of a pair of mirrors in the presence of a feed horn, and can output a few antenna patterns. The input file might thus contain the name of a file describing the feed horn, the names of two files describing the mirrors, and a few parameters that describe the antenna patterns to be generated. We can create a simple Tcl/Tk widget that consists of an entry box, and a Browse button that uses the built-in Tcl/Tk call `tk_getOpenFile` that creates a file browser with which the user can select a file, and have that file name (and path) returned into the entry box. An example widget of this type is shown in figure 3, and the Unix version of the browse window is shown in figure 4. We can create a window that contains three of these widgets, and a number of other entry widgets, one for each parameter that is required to control the antenna patterns (perhaps Theta_start, Theta_stop, Theta_increment, Phi_state, Phi_stop, and Phi_increment.) Then we can create a button called "Run", and attach to it a little Tcl code that takes the data from the entry boxes, writes this information into a file, and then starts the analysis job using the Tcl `exec` command. That's it. One of the nice things about Tcl/Tk is that this can easily be done step by step. First, we can create Tcl code that reads keyboard input, creates an analysis code's input file, and runs that code. Then we can create a Tk program that doesn't yet do anything, except look correct. By this I mean that it has three entry boxes for file names, a number of browse buttons, some entry boxes for the antenna pattern controls, and a run button. Then we can start putting these two codes together, one step at a time, testing each step as we create it. Once all this is working, we could create a new GUI that allows us to view the output files, and then we could combine the two GUIs, so that the new GUI would set up a run, start it, and then provide the user tools to view the output. This illustrates one of the best features of Tcl/Tk: the ability to make incremental changes and to combine many pieces into one GUI.

Figure 2. The Tcl/Tk hello window. On most systems, the window will be surrounded by some decorations created by the window manager. These decorations will vary by system.

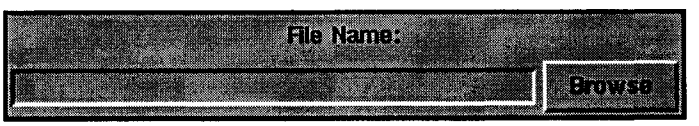

Figure 3. A sample widget for selecting file names

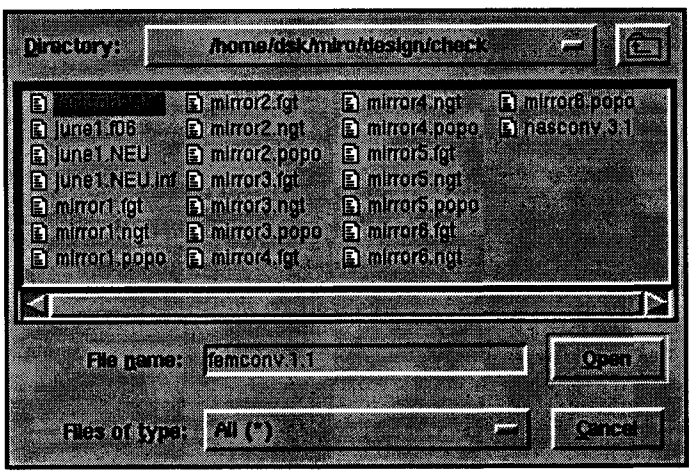

Figure 4. The Unix `tk_getOpenFile` widget

Introducing Expect

Expect was created in 1990 by Don Libes, at the National Institute of Standards (NIST) It is used for automation of interactive programs. Basically, Expect can pretend to be an interactive user for a given application. The initial ideas for Expect came from the creation of a program used to automate the initial login and command in a telnet session.

Expect is also delivered with two interpreters. One is `expect`, which is built on top of `tclsh`. In many situations, one might wish to use Expect with Tk graphics. This is the purpose of the other interpreter, `Expectk`. It is a version of Expect built on top of `wish`. Expect is freely available from the Expect website listed at the end of this column.

Let's try an Expect example. This example will only work as written on a Unix machine. On Windows and Macintosh machines, it is not as simple, but the details may be found in the "Exploring Expect" book, which is discussed later. Type the following into a plain text file called `exscript`:

```
set address \
  "replace_with_your_e-mail_address"
spawn ftp ftp.uu.net
expect "Name"
exp_send "ftp\r"
expect "Password:"
exp_send "$address\r"
expect "ftp> "
exp_send "cd pub\r"
interact
exit
```

Now, type "`expect exscript`". This will cause expect to start, and then to follow the commands in the script, which will cause it to store your e-mail address in a Tcl variable called address, start up an ftp command to ftp.uu.net, wait for the ftp command to prompt for your username, then send the string "ftp" followed by a return, wait for ftp to send a password prompt, then send your e-mail address, wait for an ftp prompt, then change directories to /pub, and then allow you to interact with the ftp program directly. When you quit the ftp program, the control will return to the expect script, where the exit statement will cause expect to exit, returning control to you. What happens if no connection can be made to ftp.uu.net? Or if this machine does not allow anonymous ftp? The script can be written to be much more clever in order to handle these possibilities, but this is beyond the scope of this column.

A GUI for an Interactive Code

As an example, let us assume that a finite-difference time-domain (FDTD) code exists which is interactive. This code provides a prompt "`fdtd>` ", waits for the user to type a command, performs that command, then issues another prompt. Sample commands are:

```
new_geom new_geometry_file_name
timestep number_of_timesteps
trace x,y,z,tp
far_fields th1,th2,dth,ph1,ph2,dph
exit
```

A code of this type is useful when the late-time convergence of a problem is unknown, as the user can examine far-field patterns at a number of time steps and interactively decide when the code has converged.

As the above Expect example demonstrated, this would be easy to automate, if we could write a script that would store the options to each command in Tcl variables. However, as we don't know these at the time we write the code, we can use Tk widgets to allow the user to enter these variables. A partial implementation of a GUI for this code might have start and stop buttons, to start and stop the FDTD code, and then a row of widgets for each option, with a button to execute that option. The Expectk script attached to each button would build its command by reading the widgets on it's row, and using `exp_send` to send that command to the FDTD program. It could read any returned information using Expect, and would wait for the FDTD program's prompt before accepting more actions from the user. Someone that has been working with Tcl/Tk and Expect for a short time could write this type of GUI in a few days, and an experienced programmed could probably write it in a few hours.

More Information

A large amount of additional information on Tcl/Tk and Expect is available. A few books that I have found particularly useful are: *Tcl and the Tk Toolkit*, John Ousterhout, Addison-Wesley Publishing, 1994, *Practical Programming in Tcl and Tk*, Brent Mead, Prentice-Hall, 1997, *Effective Tcl/Tk Programming*, Michael McLennan, Addison-Wesley Publishing, 1998, and *Exploring Expect*, Don Libes, O'Reilly & Associates, 1996. Tcl/Tk and Expect information is also available on the web. Two good starting points are http://www.tclconsortium.org/, and http://expect.nist.gov/. Additionally, the newsgroup comp.lang.tcl is a good place to read about some common (and uncommon) problems, and to attempt to find answers to questions that are not answered in the web sites. Additionally, all the codes and links listed in this column are available at the web site: http://emlib.jpl.nasa.gov/EMLIB/GUI/.

The research described in this column was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial or non-commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Comments are welcome at the e-mail address given on the first page.